Minimum Spanning Trees:
- Given a connected, weighted graph \( G = (V, E) \)
  \( (u, v) \in E \rightarrow w(u, v) = \text{weight} \)

- Spanning Tree:
  - A subset \( T \subseteq E \) of edges that connect all the vertices and is acyclic

Total weight:
\[
\sum_{(u,v) \in T} w(u,v)
\]

Minimum Spanning Tree (MST)
- A spanning tree of min. weight

- Euclidean Min. Spanning Tree (I)

How are data structures used?
- Transaction/Query:
  - Insert new student
    - name = “Mary” ID = 1234...
  - Closest coffee to my location

- Algorithms:
  - Dijkstra - Fibonacci Heap
  - Kruskal - Union/Find

Data Structures & Algorithm Design:
- Euclidean Min. Spanning Tree (I)

Transaction/Query:
- Add the lightest edge that causes no cycle
- Kruskal’s, Prims’, Borůvka’s

Algorithm for MSTs:
- Based on greedy construction

Lemma: Given any cut \((S, P \setminus S)\)
  - Always safe to add lightest edge \((p_i, p_s) \in E \) where \( p_i \in S \) and \( p_s \in P \setminus S \)

Applications:
- Clustering (Machine Learning)
- Approximation (TSP)
- Networking

Euclidean Mst (EMST)
- The MST of \( \mathbb{P} \)

Euclidean Graph:
Given a set \( \mathbb{P} = \{ p_1, \ldots, p_n \} \) of pts in \( \mathbb{R}^d \), this is a complete graph (all \( \binom{n}{2} \) edges)

where:
\[
w(p_i, p_j) = d(x_i, x_j) \quad \text{dist}(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

Facts:
- If \( G \) has \( n \) vertices, any spanning tree has \( n - 1 \) edges
Finding next edge?
- Brute force: $O(n^2) \Rightarrow O(n^3) \bigcirc$
- kd-tree: To compute near neighbor
- Priority queue: To find best pair

Nearest-Neighbor Pairs:
Given $p_i \in S$, let $p_j$ be the closest point in $P \setminus S$.
($p_i, p_j$) is nearest-neighbor pair

Prim's Algorithm:
- Given point set $P$ + start pt $s_0$
- $S \subseteq P$: Pts in spanning tree
  Init: $S = \{ s_0 \}$ End: $S = P$
- $P \setminus S$: Pts not yet in tree
  while ($S \neq P$)
    - Find closest $(p_i, p_j)$ $\Rightarrow$ $p_i \in P \setminus S$
    - add $p_i$ to $S$
    - add (p_i, p_j) to tree

Basic Objects:
- edgeList: list of edges in tree
- inEMST: set representing $S$
- kdtree, heap: ... dependents: dep lists for all $P \setminus S$

Priority Queue: Stores the NN pairs ordered by squared dist.

List: Store edges of tree (e.g. $\{(SFO,DFW),(DFW,ORD),..\}$)
Set: Store points of $S$ (e.g. $\{SFO,DFW,ORD,ATL\}$)

Spatial Index: Stores pts of $P \setminus S$. Answers NN queries

Hash map of lists: Stores dependency lists, indexed by point

How to do this?
- Lots of data structures!
addEdge(Pair(Point) edge)
add edge to edgeList's (first, second)
pt2 <- edge.getSecond()
add pt2 to in EMST
delete pt2 from kdTree
dep2 <- get pt2 dep list from dependents
for each (pt3 in dep2)
    nn3 <- kdTree.nearNeigh(pt3)
    if (nn3 == null) break
    add NN(pt3, nn3)

Q: Why check nn3 == null?
    - On adding last pt to EMST
      the kd-tree is empty.

addNN(Point pt, Point nn)
dist <- distancesq(pt, nn)
pair <- new Pair(pt, nn)
insert pair in heap w. priority dist
add pt to dep[nn]

initialize(Point start)
clear: edgelist in EMST
heap + kdTree
for each (dep in dependents)
    clear dep
    for each(pt in P)
        if (pt ≠ start) insert pt in kdTree

Helpers:
- Initialize(Point start)
  - initialize all structures
  - addEdge(Pair(Point) edge)
  - add new edge to EMST
  - addNN(Point pt, Point nn)
  - add new NN pair (pt, nn)

Euclidean MSTs (III)

That's it!

Is this efficient?
- Assuming NN queries in $O(\log n)$ time
Total time = $O(n\log n + m\log n)$

$ m $ = # of NN updates

⇒ Much depends on $ m $
  $ m $ depends on pt. distr.